(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 684 558 A1**

(12) EUROPEAN PATENT APPLICATION

(21) Application number: 95302184.7

(22) Date of filing: 31.03.95

(51) Int. Cl.6: G06F 11/14

(30) Priority: 23.05.94 US 247422

(43) Date of publication of application:
29.11.95 Bulletin 95/48

(84) Designated Contracting States:
DE FR GB

(71) Applicant: International Business Machines Corporation
Old Orchard Road
Armonk, N.Y. 10504 (US)

(72) Inventor: Shaheen, Amal Ahmed
6321 Pathfinder Drive
Austin,
Texas 78759 (US)
Inventor: Yellepeddy, Krishna Kishore
13026 Partridge Bend Drive
Austin,
Texas 78729 (US)

(74) Representative: Davies, Simon Robert
I B M
UK Intellectual Property Department
Hursley Park
Winchester, Hampshire SO21 2JN (GB)

(54) Distributed data processing system.

(57) A system and method are provided for maintaining data coherency in a system 100 in which data is replicated on two or more servers 102, 104. Each server is able to update the data replica present on the server. Updates are logged for each server. Reconciliation of server data replicas is aggressively initiated upon the occurrence of predefined events. These events include arrival at a scheduled time, a request for data by a client system, server and network failure recovery. Reconciliation is managed by a coordinator server selected to ensure that at most one coordinator server per network partition is selected. Logged updates are merged and transmitted to each server containing a data replica. The logged updates are applied unless a conflict is detected. Conflicts are collected and distributed for resolution. Reconciliation is managed between servers without regard to operating system or physical file system type.

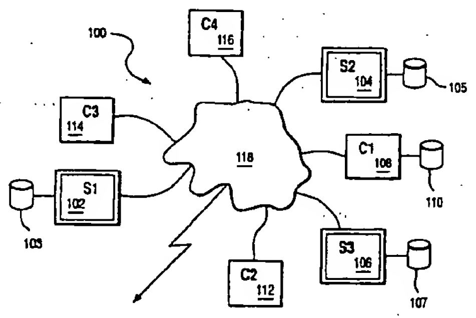

FIG. 1

T1

The present invention relates to a distributed data processing system, and more particularly to the data management of replicated data in a distributed processing system.

Computer systems can consist of one or more computers. Distributed computer systems are created by linking a number of computer systems by a private communication mechanism, local area network (LAN) or wide area network (WAN). Each of the linked computers typically has a processor, I/O devices, volatile storage, and non-volatile storage. Certain ones of the computers may be designated as "servers." A server provides services for one or more other computers which are labelled "clients." A server usually provides non-volatile (e.g. hard disk) storage that can be shared by a number of computers. Servers may also provided shared processing resources and shared access to expensive peripherals, such as high speed printers or scanners.

The sharing of data resident on a server has certain advantages for departments or workgroups with common data processing requirements. Distribution of data over a network, however, can create problems if the network or server experiences an outage. In addition, access to data over a wide area network may be costly and may have increased access times.

One approach to increasing the availability and reliability of distributed systems is to replicate data on more than one server. Thus, a particular shared data file may be replicated or copied to one or more other servers. As a result, a client may recover from failure of a server or network segment by accessing a different replica.

Replication policy can be either "optimistic" or "pessimistic." A pessimistic replication policy requires that at most one copy of the data be writable or updatable. This ensures data consistency because only one copy of the data can be changed and that copy is always assumed to be correct. Other replicas under the pessimistic policy are designated "read-only."

The pessimistic policy ensures data integrity but may result in limited availability of the updatable replica to any one of the clients. An optimistic replication policy, in contrast, allows multiple replicas to be concurrently updatable. Conflicting updates are detected and resolved after they occur. An optimistic replication policy provides higher write availability to client systems. Write sharing between users, i.e. multiple users updating multiple replicas of the same data, is assumed to be infrequent and, consequently, the impact of potential conflicts is minimized.

An optimistic replication policy, however, can lead to inconsistencies in data between replicas on different servers due to server or network failure. A technical problem exists to manage replica updates to minimize inconsistencies and to detect and resolve any inconsistencies in a timely manner.

Existing distributed file systems have been unable to provide a satisfactory solution to this problem. The Andrew File System (AFS) from Transarc Corp. and the Distributed File System (DFS) for the Distributed Computing Environment (DCE) from the Open Software Foundation (OSF) each implement replicated data with a pessimistic replication policy. AFS and DFS allow at most one updatable replica, thereby avoiding the data conflict problem, but sacrificing availability.

The CODA distributed file system, a research project developed at Carnegie Mellon University, implements an optimistic replication policy (see M. Satyanarayanan et al, "CODA: A Highly Available File System for a Distributed Workstation Environment", IEEE Transactions on Computers, Vol. 39, No. 4, April 1990.) A client system data request initiates replica update for the data. This update policy has the disadvantage of being controlled by the client, which is assumed to be untrustworthy. It is also not timely because no conflict detection occurs unless a client requests particular data. The CODA approach may also prove to be costly in cases where the communications cost for client-to-server connections is high.

Lotus Notes allows multiple read/write replicas of its special database. Replicas are periodically reconciled, usually no more than once or twice a day. Detection of a conflict between replicas causes the creation of separate versions with no attempt to automatically resolve the conflicts. This results in a significant manual management burden whenever a conflict occurs.

In a paper by Tait et al [see "Server Interface and Replica Management Algorithm for Mobile File System Clients", Computer Science Department, Columbia University], the problem of maintaining replicas in a distributed file system is investigated. The authors argue that in an environment with mobile file system clients a lazy server-based update operation should be used for replica management, and present a solution for this lazy server-based update.

Thus, the technical problem remains of creating an optimistic replication system that permits multiple updatable replicas but identifies and resolves conflicts in a timely and efficient manner.

Accordingly, the invention provides a distributed data processing system having a plurality of processors interconnected by a communications medium and a plurality of storage devices, each of said storage devices controlled by at least one of said processors, in which data is replicated on two or more of said plurality of storage devices, the system including means for maintaining coherency of the replicated data comprising:

means for detecting an event requiring an update of said replicated data;

means for selecting one of said processors to coordinate the update of said replicated data; and

means for updating said replicated data, said means for updating causing each data replica to become a duplicate of all other replicas.

Such an approach solves the problem of efficiently maintaining replicated data coherency or consistency in a data processing system using optimistic replication. The system servers coordinate among themselves to detect replica inconsistencies and initiate an update protocol to detect stale or conflicting replicas without waiting for a client request for data.

In a preferred embodiment, certain of said processors are designated server processors and said means for selecting selects a processor from said server processors; and said means for selecting includes: means for establishing a priority for each of said processors; means for querying each higher priority processor to determine whether it will coordinate an update; means for detecting responses to said query; and means for signalling assumption of coordination role to processors of lesser priority.

The invention further provides a method of maintaining data coherency between data replicas distributed between two or more servers connected by a network, the method comprising the steps of:

logging changes to a data replica;

detecting an event requiring reconciliation of replicas;

selecting a server to coordinate reconciliation;

determining changes needed to each replica based on said logged changes;

making the determined changes and detecting any data conflicts; and

recording data conflicts for later resolution.

In a preferred embodiment, said events include a scheduled update, the return to service of a server, and a network link reconnection; and each data replica is stored on a physical file system and said selected server maintains a physical file system that differs from the physical file system of at least one of the other servers.

It is also preferred that the step of selecting a server to coordinate reconciliation comprises the steps of:

assigning a priority to each of said two or more servers;

sending by each of said two or more servers of a message to each server of higher priority querying whether said higher priority server will become the coordinator server;

detecting higher priority server responses; and

if no response is detected, sending a message to each server of lower priority signalling assumption of coordinator server role by the sending server.

Two of the servers may be controlled by different operating systems, for example the selected server may be controlled by an operating system that differs from the operating system of at least one of the other servers. Preferably, one and only one coordinator server is selected for each system partition.

Thus a system may maintain coherency of data in a data processing system, such as a cluster or distributed processing system, having a plurality of processors interconnected by a communications medium and a plurality of storage devices, in which each of the storage devices is controlled by at least one of the processors. Data is replicated on two or more of the plurality of storage devices. The system comprises means for detecting an event requiring update of the replicated data and means for coordinating updates of the data, the means for coordinating causing each data replica to become a duplicate of all other replicas. This provides a replicated data management system with an optimistic replication policy and server controlled conflict resolution. One and only one server is selected to coordinate updates in a network where one or more servers may have failed, and more than one server requires updating by the update process. The update policy efficiently maintains data consistency and resolves most conflicts without manual intervention. The data replication policy is independent of the server operating system, the network protocol, and the physical file system organization.

An embodiment of the invention will now be described in detail by way of example only with reference to the following drawings:

Figure 1 is a block diagram illustrating a distributed data processing system;

Figure 2 is a block diagram of a typical server component of the data processing system of Figuer 1;

Figure 3 is a state diagram depicting the server states in the replication process;

Figure 4 is a flow diagram depicting the update coordinator election process;

Figure 5 is a flow diagram depicting the update process at the master coordinator server and the subordinate servers; and

Figures 6 and 7 are diagrams illustrating the layout of a modification log.

A data processing system is shown in Figure 1, the overall clustered or distributed system **100** comprising a number of clients and servers interconnected by a network **118**. Network **118** may be a local area network (LAN) serving a small business, corporate department or similar entity, or it may include wide area network components interconnecting geographically dispersed sites. Network

118 may also be a private network, e.g. a high speed fiber optic connection or a switch connecting a cluster of machines. The present invention is applicable in all these cases.

Clustered or distributed system **100** includes servers **102**, **104**, and **106**, each of which has non-volatile storage **103**, **105**, and **107**. The non-volatile storage may be magnetic hard disks, optical disks or similar technology. Each of the servers may have any number of each of these types of storage devices. Server systems typically contain large amounts of non-volatile storage having capacities exceeding one gigabyte.

Client workstations **108 112 114 116** are connected to the network and are able to use the services of one or more of servers **102 104 106**. A client may have its own non-volatile storage, (e.g. **110**) or may rely totally on a server for non-volatile storage.

The configuration of system **100**, including the number and configuration of clients and servers, is provided only as an example may vary significantly between systems.

An example of a server is shown in Figure 2 generally at **200**. A server can be any computer system such as an IBM PS/2 Model 95 or an IBM RISC System/6000. Server **200** has one or more processors or CPUs making up a processor element **202**. Processor element **202** has access to volatile memory **204**. The processor also controls Input/Output unit **206**, which controls access to non-volatile storage, such as hard disk **208**, and to input/output devices such as display **210**, keyboard **212**, and pointing device **214**. Other I/O devices could be attached to the input/output unit such as a diskette drive, a CD-ROM drive and other multimedia devices. Network communications is handled by network controller **216**. Network controller **216** performs the services necessary to control access to the network and other servers on the network. The physical network can be a token ring, Ethernet or other network. The network protocol can be TCP/IP, Netbios, Netware, or other protocol allowing access to resources and transmission of messages over a network.

The present invention may be implemented in either a homogeneous environment, or a heterogeneous environment, in which the server operating systems vary, e.g. being a mixture of AIX, OS/2 and other operating systems. The approach described below is not dependent on client or server operating system, client/server protocol, network protocol or physical file system; further, the replicated data need not be managed by a formal database management system (DBM).

The computer systems in Figure **1** are operated by stored instructions, which may be stored in non-volatile storage such as disk drive **208**, or can be stored in a removable medium. The instructions cause the server to become configured in the manner necessary to implement the following process.

Figure 3 will be used to describe the replica reconciliation process on a server. Each server in the network is in one of five states at any point in time. The five states and the transitions between them are shown in Figure 3. The server is generally in an "idle or running" state **302**. "Idle" refers to the fact that the server is not performing an update function or participating in an election. Failure of the server due to hardware or software failure causes the server to move into the "down" or failure state **301**. In the "down" state the server is unable to service client requests and is unable to participate in any election or replication update process. Once the system is restored to "idle or running" state **302** the server moves to the electing state **304**. The server may also move to the electing state when an "update event" is detected. The electing state **304** results in the selection of an update coordinator server and leads to "reorganizing" step **306** along path **316**. If no other processor participates in the electing, e.g. if they are already in a reorganizing or updating state, then the server returns to "idle or running" state **302** via path **315**. In reorganizing state **306** the coordinating server collects and merges the modification logs from the participating servers and sends the merged log back to the participating servers. Finally, each of the servers moves to "updating" state **308** in which the update is performed. Upon completion of its update, the server returns to the" idle or running" state 302. The server may go to a "down" state **301** from any one of the other states.

The preferred embodiment optimistically replicates a fileset or volume at multiple servers. A fileset can be a subtree of a file system or an information directory. The set of servers which store a replica of the fileset is called the fileset storage group (FSG). In a system with multiple servers, each fileset storage group may include different servers and each server may have a replica of any number of filesets. Each replica can be designated as read-write (i.e. updatable) or as read-only.

All operations that cause a change to the data of a read-write replica are entered in a modify log maintained in non-volatile (persistent) storage. The logged operations are known as mutating operations and include operations such as creating a directory, renaming a file, and removing a directory. A separate modify log is maintained for each fileset replica at a server. Mutating operations are routinely transmitted through the network to other servers in the fileset storage group to allow them to make the same modifications and maintain data consistency between the replicas. Network failure or remote server failure may result in the failure to execute a mutating operation at a particular remote server. This failure leads to inconsistent data.

A modification log entry, as illustrated in Fig. 6, is the basis for updating the fileset replicas to maintain data coherency. Thus, the log entry must contain sufficient information to accomplish the update. The log entry typically contains an operation independent portion and an operation dependent portion. Operation independent portion **602** contains, in the preferred embodiment, the userid of the update **606** and the date of the transaction **608**. Operation dependent portion **604** varies by operation. Figure 6 shows a store-log entry and Figure 7 a create entry. The create log entry consists of the following: the file id **710** of the directory in which the file was created, name of the file being created 712, file id of the file being created, 714; the open mode **716** of the file; open flags **718**, and file version **728**.

Replica coherency first requires identification of the servers with replicas that are missing updates and then requires transmission of the necessary updates to that server for application to the out of date replica. The preferred embodiment implements an aggressive update propagation protocol for determining when replica updates must occur. The protocol is termed aggressive because it is designed to cause replica update as soon as possible after failure without waiting for a request for data from a client system. Prior art systems use a lazy policy in which the update process is not triggered unless a client request detects stale data. The update propagation protocol triggers updates in two ways. The first is periodic updates at scheduled intervals or scheduled times. The second is the detection of system events that require replica update. Clients do not participate in the update process because a client machine is not trusted and may have limited processing capability.

Scheduled updates can be defined at a particular frequency or time. Updates can also be event-triggered. For example, a heartbeat mechanism can be used to detect network or server failure and recovery from such failure. If a server does not receive a heartbeat pulse from another server in the fileset storage group within a specified interval of time, it assumes that a network or server failure has occurred. Recovery from this failure is then used to trigger an update process. The preferred embodiment, however, does not implement a heartbeat mechanism because of the network resources such a mechanism would consume.

System events that cause update propagation include recovery from a server crash, recovery from a network partitioning due to partial network failure, and a request for data by a client. Recovery from a server failure or network failure can be detected using a heart-beat mechanism. A request for data by a client causes the server to probe the other servers for the most current copy. If the server detects that one or more of the replicas are stale, the requesting server will trigger an update. Note in the latter case the request for data by a client may trigger an update propagation, but the update will not be controlled by the client as was the case in prior art systems.

An update event causes the server to move to the "electing" state **304** which performs the election process. The election process performed in the electing state is shown in greater detail in Figure 4. The election process implements a modified "bully" algorithm, in which the server with the highest priority is elected. Each server is assigned a unique priority in the network. The preferred embodiment assigns one as the highest priority, though other priority schemes could be employed. A server is eligible to participate in the election process as long as it is not already participating in an update process for the fileset under consideration. The modification to the algorithm enables it to handle a large number of server failure conditions including server sleep and server hardware and software failure.

The process begins at step **402** and proceeds immediately to **404**. Each server in the fileset storage group broadcasts a message to all other higher priority servers in the fileset storage group every <u>T</u> seconds. If a server, <u>B</u>, does not receive a response from any of the higher priority servers within a specified time period **406**, then it assumes that all higher priority servers have failed and that it can be the coordinator. Server <u>B</u> then sends a message **410** to all servers having a lower priority indicating that it is the coordinator for updates to a specific fileset and then coordinates the update process **412**.

If one or more servers with a higher priority than <u>B</u> respond that <u>B</u> cannot be the coordinator, then <u>B</u> waits for a specified time **414** for a message from a higher priority server that it has become the update coordinator. If no message is received **416**, then server <u>B</u> assumes all higher priority processors have failed and it becomes coordinator, sends a coordinator message to lower priority servers **410**, and coordinates the update process **412**.

The preferred embodiment uses a modified "bully" algorithm to handle failure conditions. The modified algorithm ensures that each server will have only one coordinator, but does not guarantee that only one coordinator will be elected. The modified algorithm recognizes the current state of each server in the fileset storage group (as shown in Figure 3.) The state can be:

down:	server will not respond to election announcements or locking requests for a fileset update protocol;
idle:	server is idle and between fileset updates;
electing:	server is currently participating in an election, i.e. it is waiting to hear from servers with higher priority whether or not it can become coordinator;
reorganizing:	server is currently involved in a lock and fetch operation, i.e. the fileset is write locked and the fileset's update log is extracted from persistent storage;
updating:	server is participating in an update.

A server that is reorganizing, updating or down does not participate in an election.

The election of a coordinator is followed by the update process. The remaining update phases are shown in Figure 5. The processes for the coordinator and subordinates are both shown to help describe the interaction. The process starts **502 504** with the election of a coordinator server. The coordinator server first locks its own fileset and sends a request to all subordinates to lock that fileset and to send the current fileset version vector to the coordinator.

Subordinates that successfully lock the fileset become subordinates for the rest of the update protocol. If the coordinator receives no acknowledgement from any subordinate that it is able to lock the fileset, or if all subordinates respond that the fileset is already locked, then the coordinator state for this server ends and a new election is initiated after a specified delay.

A fileset version vector is maintained by each server for each fileset replica [see Parker et al, "Detection of Mutual Inconsistency in Distributed Systems"", IEEE Transactions of Software Engineering, May 1983]. Each modification to a replica on a server is tagged with a unique store-id by the client performing the operation. The server maintains an approximation of the update history for a replica consisting of the latest store-id and length of update history, i.e. the number of updates to this replica. The server also maintains an estimate of the length of the update history of every other replica in the fileset storage group. The vector of update lengths is the fileset version vector.

Two replica states can be compared using their store-ids and version vectors. The result indicates whether the replicas are identical, inconsistent due to concurrent updates to both replicas, or unequal (one replica updated but not the other.)

The coordinator compares replica states. If all replicas are identical, no further action is required and the update process ends.

Next the coordinator requests the modification log from each subordinate **506** for the fileset to be updated. Each subordinate sends its modification log to the coordinator **508**. A subordinate who fails to respond within a specified time is dropped from the update.

The coordinator merges the logs **510** from all subordinates and sends **512** the merged log to all subordinates. Each subordinate performs the updates **514** that were missing from its replica. In the preferred embodiment, the missing updates are executed within a single transaction to guarantee atomicity. Application of the update is noted in the modification log. Once all updates are applied, the changes are committed and the replay ends.

If the subordinate cannot replay an update due to conflict it is added to a conflict list that is sent **516** to the coordinator after all updates are attempted. Concurrent updates made to a file or directory in multiple partitions which cannot be merged without violating semantic constraints are called conflicting. Concurrent updates of a directory which do not refer to the same object can be merged automatically as they are not conflicting. Conflict resolution specific to a particular application such as bulletin boards or e-mail can also be provided to allow automatic conflict resolution.

Once the updates are complete, the coordinator informs the subordinates that the fileset may be unlocked. If the update phase completes successfully, the coordinator sends a new store-id to mark all replicas as equal. If the update replay phase was not successfully completed due to conflicts, the coordinator determines **518** which files are in conflict and marks them as such. A conflict list is sent to each subordinate **520**. A client may access the files on the conflict list using a repair utility to manually repair the conflicting files.

The preferred embodiment uses a heartbeat mechanism to detect coordinator or subordinate failure during the update process. The coordinator periodically, e.g. every h seconds, sends a heartbeat pulse to each subordinate. If a subordinate does not receive a pulse for 2h seconds, it concludes that the coordinator has failed. The coordinator similarly detects failure of a subordinate if a heart-beat signal is not received from the subordinate for 2h seconds.

Claims

1. A distributed data processing system (100) having a plurality of processors (102, 104, 114) interconnected by a communications medium (118) and a plurality of storage devices (103,

105), each of said storage devices controlled by at least one of said processors, in which data is replicated on two or more of said plurality of storage devices, the system including means for maintaining coherency of the replicated data comprising:
means for detecting an event requiring an update of said replicated data;
means for selecting one of said processors to coordinate the update of said replicated data; and
means for updating said replicated data, said means for updating causing each data replica to become a duplicate of all other replicas.

2. The system of claim 1, wherein certain of said processors are designated server processors and wherein said means for selecting selects a processor from said server processors.

3. The system of claim 1 or 2, wherein said means for selecting includes:
means for establishing a priority for each of said processors;
means for querying each higher priority processor to determine whether it will coordinate an update;
means for detecting responses to said query; and
means for signalling assumption of coordination role to processors of lesser priority.

4. A method of maintaining data coherency between data replicas distributed between two or more servers connected by a network, the method comprising the steps of:
logging changes to a data replica;
detecting an event requiring reconciliation of replicas;
selecting a server to coordinate reconciliation;
determining changes needed to each replica based on said logged changes;
making the determined changes and detecting any data conflicts; and
recording data conflicts for later resolution.

5. The method of claim 4, wherein said events include a scheduled update, the return to service of a server, and a network link reconnection.

6. The method of claim 4 or 5, wherein each data replica is stored on a physical file system and said selected server maintains a physical file system that differs from the physical file system of at least one of the other servers.

7. The method of any of claims 4 to 6, wherein the step of selecting a server to coordinate reconciliation comprises the steps of:
assigning a priority to each of said two or more servers;
sending by each of said two or more servers of a message to each server of higher priority querying whether said higher priority server will become the coordinator server;
detecting higher priority server responses; and
if no response is detected, sending a message to each server of lower priority signalling assumption of coordinator server role by the sending server.

8. The method of claim 7, further comprising the step of detecting coordinator server failure and causing the step of selecting a server to be repeated.

9. A method of operating a system having a plurality of servers and data replicated at two or more of said servers to select a coordinator server to coordinate data replica updating, the method comprising the steps of:
assigning a priority to each of said servers;
sending by each of said servers of a message to each server of higher priority querying whether said higher priority server will become the coordinator server;
detecting higher priority server responses; and
if no response is detected, sending a message to each server of lower priority signalling assumption of coordinator server role by the sending server.

10. The method of any of claims 7 to 9, wherein one and only one coordinator server is selected for each system partition.

11. The method of any of claims 7 to 10, wherein said selected server is controlled by an operating system that differs from the operating system of at least one of the other servers.

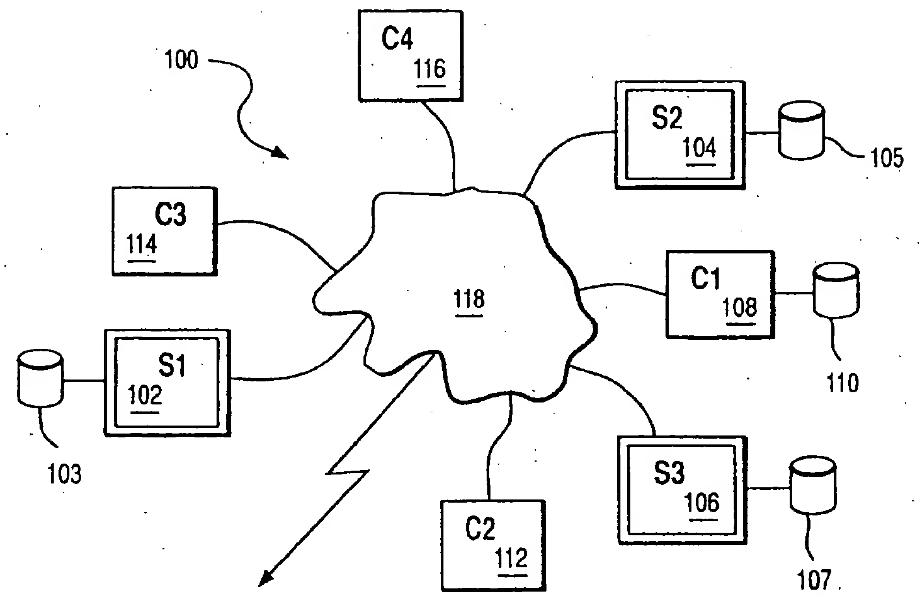
100
C4
116
S2
104
105
C3
114
118
C1
108
110
S1
102
103
S3
106
107
C2
112

FIG. 1

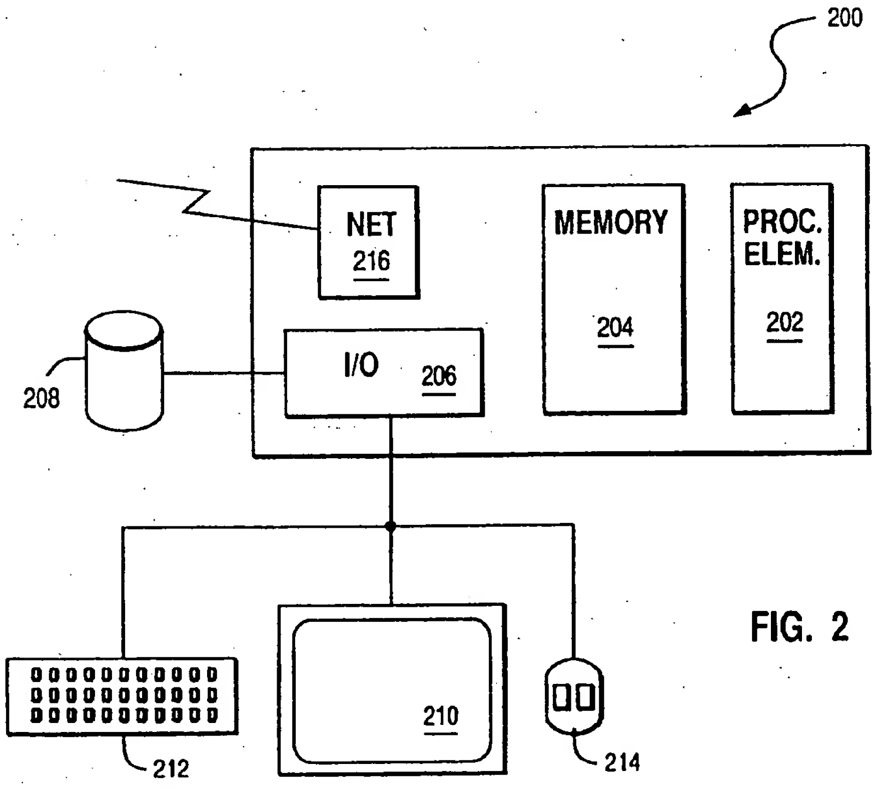
200
NET
216
MEMORY
204
PROC. ELEM.
202
I/O
206
208
210
212
214

FIG. 2

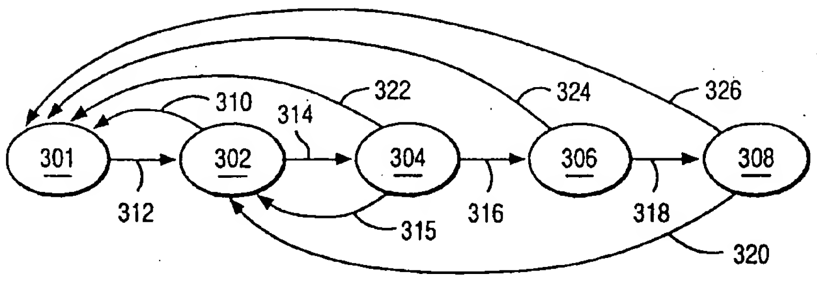
301
302
304
306
308
310
312
314
315
316
318
320
322
324
326

FIG. 3

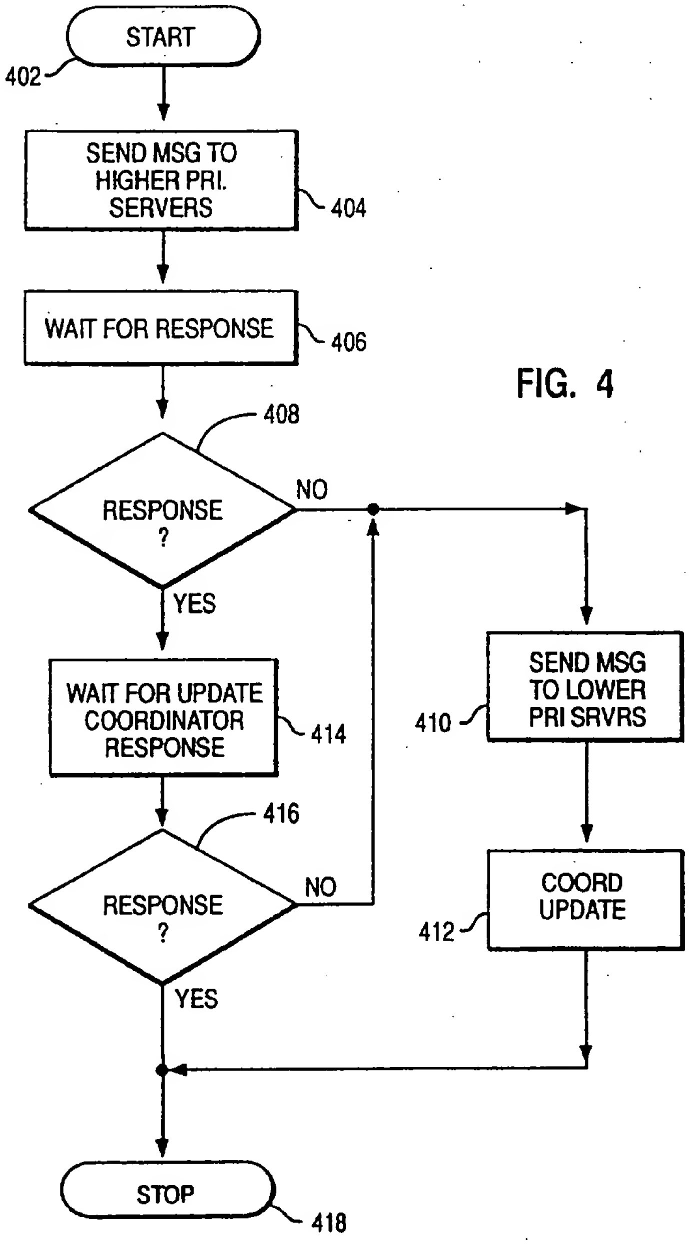
START
402
SEND MSG TO HIGHER PRI. SERVERS
404
WAIT FOR RESPONSE
406
408
RESPONSE ?
NO
YES
WAIT FOR UPDATE COORDINATOR RESPONSE
414
416
RESPONSE ?
NO
YES
SEND MSG TO LOWER PRI SRVRS
410
COORD UPDATE
412
STOP
418

FIG. 4

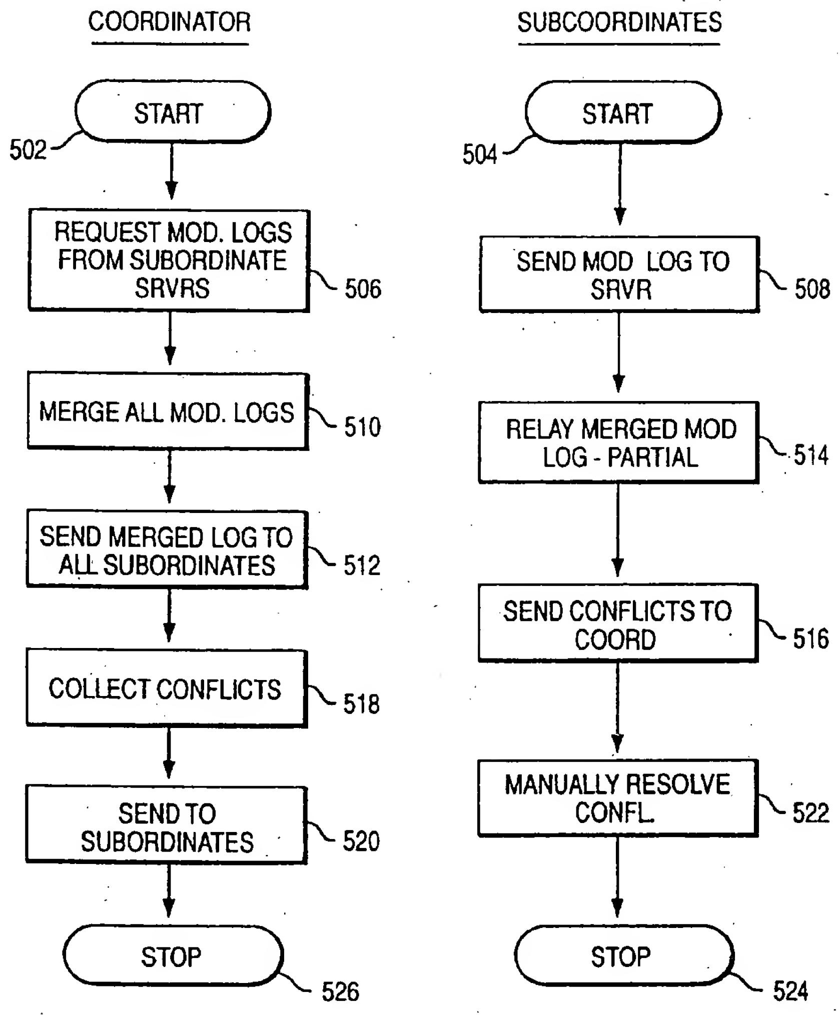
COORDINATOR
SUBCOORDINATES
START
502
START
504
REQUEST MOD. LOGS FROM SUBORDINATE SRVRS
506
SEND MOD LOG TO SRVR
508
MERGE ALL MOD. LOGS
510
RELAY MERGED MOD LOG - PARTIAL
514
SEND MERGED LOG TO ALL SUBORDINATES
512
SEND CONFLICTS TO COORD
516
COLLECT CONFLICTS
518
MANUALLY RESOLVE CONFL.
522
SEND TO SUBORDINATES
520
STOP
526
STOP
524

FIG. 5

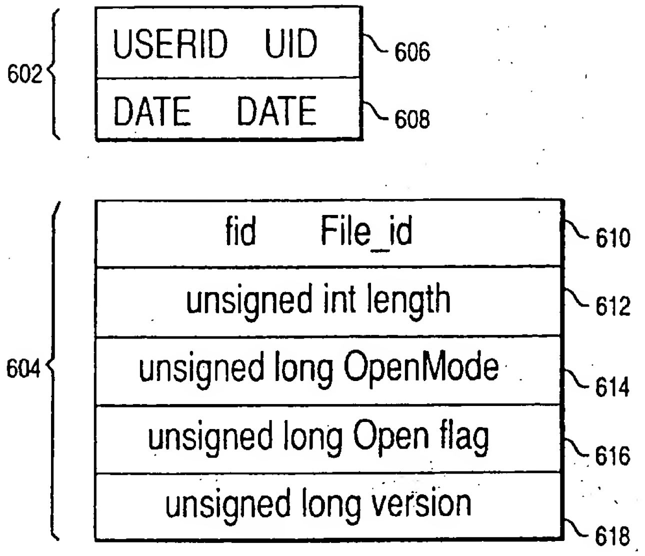
602
USERID UID
DATE DATE
606
608
604
fid File_id
unsigned int length
unsigned long OpenMode
unsigned long Open flag
unsigned long version
610
612
614
616
618

FIG. 6

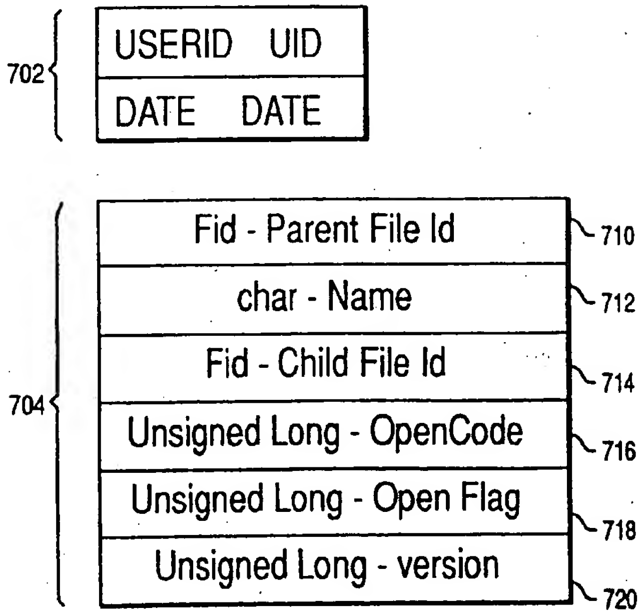
702
USERID UID
DATE DATE
704
Fid - Parent File Id
char - Name
Fid - Child File Id
Unsigned Long - OpenCode
Unsigned Long - Open Flag
Unsigned Long - version
710
712
714
716
718
720

FIG. 7

European Patent Office

EUROPEAN SEARCH REPORT

Application Number
EP 95 30 2184

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.6)
X	IBM TECHNICAL DISCLOSURE BULLETIN, vol.36, no.11, November 1993, NEW YORK US pages 675 - 678 'Bounding Journal Back-Off during Recovery of Data Base Replica in Fault-Tolerant Clusters' * page 677, line 4 - line 10 * ---	1,2,4	G06F11/14
A	IBM TECHNICAL DISCLOSURE BULLETIN, vol.35, no.3, August 1992, NEW YORK US pages 286 - 289 'Centralized and rapid backup/restore for Work LAN File Services/VM' * page 287, line 3 - line 7 * -----	3,6	

TECHNICAL FIELDS SEARCHED (Int.Cl.6)
G06F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
THE HAGUE	18 August 1995	Corremans, G

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document